Licenciatura Engenharia Informática e Multimédia

Redes de Computadores

Semestre de Verão 2022 / 2023

Trabalho Prático 1

Docente Luís Pires

07 de Abril de 2023

Trabalho realizado por:

Fábio Dias, nº 42921 Grupo 17

Índice

Índice de Figuras

Lista de Acrónimos

HTTP – HyperText Transfer Protocol

IP – Internet Protocol

PC – Personal Computer

RFC – Request For Comments

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

1. Introdução

Para a primeira fase do trabalho foi-nos pedido para instalar um Servidor Web no PC e aceder ao mesmo a partir de outro dispositivo, usando o nosso endereço IP, e usando uma aplicação desenvolvida por nós. Desenvolvendo, assim, o modelo Cliente-Servidor[1].

O objetivo é analisar a troca de mensagens, com o auxílio do WireShark[2], com o protocolo HTTP[1, 3, 4] entre cada Cliente e o Servidor.

O objetivo desta primeira fase é consolidar o nosso conhecimento, tanto no protocolo HTTP, assim como fundamentar como é que a comunicação entre duas máquinas distintas e separadas é estabelecida, sem perder dados e, realmente, receber, enviar e tratar mensagens de ambas as partes, assim como fundamentar o nosso conhecimento nas diferentes camadas existentes do modelo em que as comunicações em redes são baseadas, que será apresentado neste relatório.

2. Layered Internet Protocol Stack

Dado que um sistema de redes é complexo, devido a aplicações, protocolos, tanto o hardware como o software, podemos organizá-lo de forma a obter uma estrutura organizada. Para este objetivo, observamos esta estrutura desenvolvida por "camadas".

Cada "camada" possui uma identificação e uma relação com as camadas adjacentes, tanto superiores como inferiores, caso existam. Desta forma conseguimos modularizar esta estrutura, o que facilita a sua atualização e manutenção. Se for preciso atualizar ou alterar qualquer coisa nesta estrutura, não precisamos de nos focar em cada camada. Basta identificarmos qual a camada objetivo e prosseguimos com o processo, sem nunca ter de modificar o que quer que seja nas outras camadas, adjacentes ou não.

Esta estrutura já foi pensada e desenvolvida e possui o nome Layered Internet Protocol Stack[1, 5]. O seu modelo de referência foi o Modelo OSI (Open Sysems Interconnection) com algumas alterações feitas, em particular, a integração de duas "camadas" numa das camadas já existentes. Isto porque a sua funcionalidade, neste contexto, não era distinta o suficiente para possuírem camadas próprias, ou seja, não eram assim tão específicas para este modelo ao ponto de terem as suas camadas individuais.

Assim sendo, este modelo apresenta a Camada de Aplicação, onde se encontra a aplicação a correr. É aqui que, escondido do utilizador, existe suporte para os protocolos da Camada de Rede. De seguida, vem a Camada de Transporte. É nesta que funcionam os protocolos TCP[1, 6] e UDP e onde os pacotes de dados a serem enviados são processados. A próxima camada já foi mencionada. A Camada de Rede. É nesta camada que os pacotes são encaminhados para os IP's e Portos corretos. Estes pacotes passam pela Camada de Ligação de Dados, que passam os pacotes de dados já preparados para os meios que os vão enviar. Por fim, temos a Camada Física que é responsável pelo envio dos pacotes, em si, a partir de um meio de comunicação físico. (Figura 1)

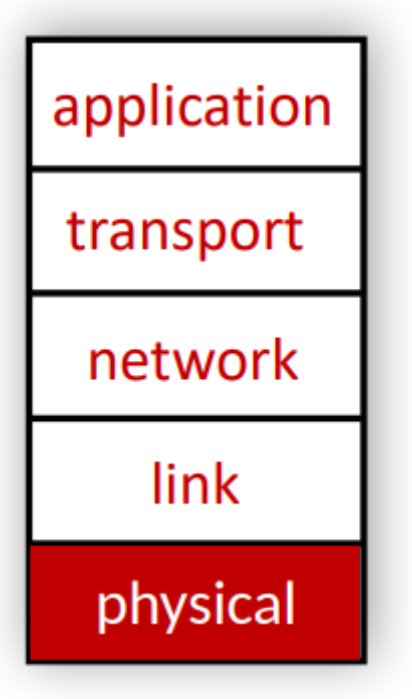

Figura 1 - Layered Internet Protocol Stack

Qualquer troca de mensagens, seja que protocolo for, que seja gerada pela Camada de Aplicação, ao passar para a Camada de Transporte, é encapsulada com o protocolo de transporte pretendido e, assim, esta mensagem passa a ser um segmento. Ao chegar à Camada de Rede, esta é encapsulada com o protocolo de rede e, assim, gera um datagrama. Por sua vez, este datagrama, ao passar para a Camada de Ligação de Dados é encapsulada com o seu cabeçalho, o protocolo de ligação, e resulta numa trama. Esta, quando chega à Camada Física, é enviada para a outra máquina, seja esta qual for.

A todo este processo é chamado de Encapsulamento.

Quando a trama chega ao destinatário, esta faz o caminho inverso e, sempre que passar por uma camada, o seu cabeçalho específico dessa camada é interpretado e retirado da mensagem. Quando chega à Camada de Aplicação, a mensagem original é obtida.

3. Protocolo HTTP

O protocolo HTTP[1, 3, 4] (Hypertext Transfer Protocol) é o protocolo definido para a comunicação de aplicações Web. Está construído no modelo Cliente-Servidor, onde o Cliente envia pedidos e o Servidor recebe-os, interpreta-os e envia uma resposta que o Cliente é capaz de interpretar.

Este protocolo usa o protocolo TCP, a partir de sockets, para o envio de dados. O cliente inicia sempre a conexão com o Servidor, no porto 80. O Servidor, por sua vez, aceita a conexão e existe uma troca de mensagens HTTP e, por fim, a conexão é terminada. Como mencionado, esta comunicação é feita a partir de sockets. Estes são como pontos de ligação existentes tanto no Cliente como no Servidor.

Este protocolo é "*stateless*" ou seja, não guarda qualquer informação dos pedidos anteriores do Cliente. A sua ligação pode ser não-persistente, onde cada objeto requisitado cria uma ligação TCP, existindo múltiplas ligações paralelas Cliente-Servidor, ou pode ser persistente, onde a ligação permanece após o envio das respostas, existindo apenas uma ligação para todos os pedidos.

Existem dois tipos de mensagens que, pelo que já foi explicado, é possível inferir: pedidos e respostas.

O formato geral das mensagens de pedido são constituídas por uma linha de pedido (request line), seguido de linha de cabeçalhos (header lines), adicionar uma linha em branco e, por fim, o corpo da mensagem (body). Estas mensagens podem ser de vários tipos, alguns deles são o POST, para enviar dados; GET, com o fim de obter dados; HEAD, pede apenas a linha de cabeçalhos da mensagem de resposta; PUT, que substitui dados já existentes pelos enviados.

A mensagens de resposta são constituídas por código de status (status code), seguido de linha de cabeçalhos (header lines) e, por fim, os dados (data). Estas mensagens também podem ser de vários tipos, alguns deles são o 200 OK, o pedido foi bem-sucedido e os dados requisitados serão enviados no corpo; 302 Found, isto indica que o recurso requisitado foi movido temporariamente; 400 Bad Request, indica que o pedido está mal construído ou que não conseguiu ser entendido pelo Servidor; 404 Not Found, o recurso pedido não foi encontrado; 505 HTTP Version Not Supported, isto indica que a versão do protocolo HTTP não é compatível com o do Servidor.

O protocolo TCP (Transfer Control Protocol) é um protocolo de comunicação entre computadores em rede. Este garante que tanto o Servidor como o Cliente recebem todos os dados pretendidos, por auxílio de reconhecimento (Acknowledge, ACK) de ambas as partes. Caso estes pacotes de dados não sejam recebidos e, por sua vez, reconhecidos, os pacotes de dados são enviados novamente, evitando assim, a perda dos mesmos.

Esta comunicação tem de ser começada com um "*handshake*". É aqui que muitas variáveis são concordadas e, após o reconhecimento e sincronismo de ambas as partes, esta comunicação fica estabelecida até ser fechada.

4. Servidor Web Apache

Foi usado um servidor Apache[7], via distribuição XAMPP, de forma a obtermos compatibilidade com o protocolo HTTP.

Para o instalarmos, devemos correr o seu instalador, o que nos leva ao seu menu inicial de instalação. Pressionamos "*Next*" e passamos à seleção dos componentes a serem instalados. No meu caso, deixei tudo como se encontrava, mas era apenas necessário o componente Apache. Selecionei "*Next*" e somos direcionados para o menu para especificarmos a pasta onde queremos instalar o Servidor. Seguindo para o próximo menu, escolhemos a linguagem da aplicação. Por fim, obtemos o menu que nos indica que o Servidor está pronto a ser instalado. Ao pressionarmos "*Next*" o Servidor começa a ser instalado.

Quando todo este processo estiver concluído, podemos aceder ao Painel de Controlo do Servidor, onde podemos configurá-lo, assim como ligá-lo e desligá-lo. Ao ligarmos o Servidor, este oferece-nos a informação dos portos a que nos devemos ligar. (Figura 2)

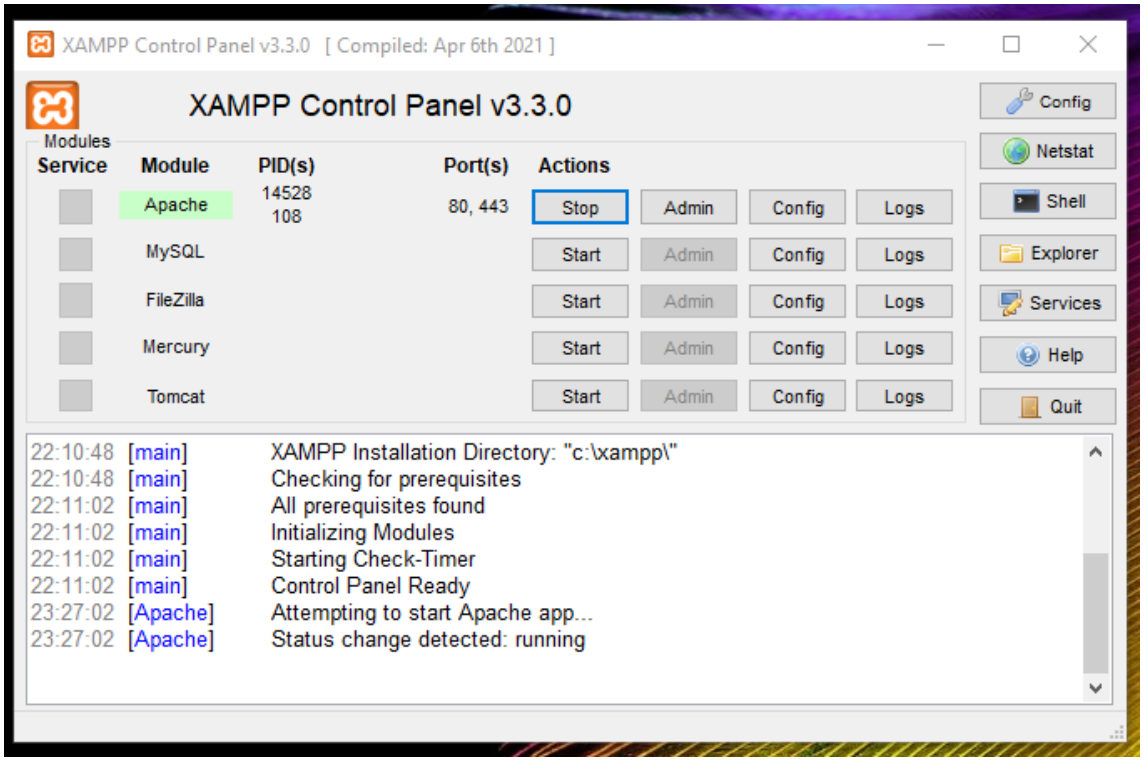

Figura 2 – Servidor Apache, Painel de Controlo

Para testarmos se o Servidor está de facto a funcionar, podemos aceder ao mesmo, por um browser à nossa escolha, e podemos usar ou o "localhost" ou o IP "127.0.0.1" na barra de endereço. (Figura 3)

Figura 3 - Servidor Apache, Página Inicial do Servidor, no Browser

5. Wireshark

Como indicado previamente, para analisar a troca de mensagens entre o outro dispositivo e o Servidor Web, vamos ter o auxílio do software Wireshark[2]. Este é usado para analisar o tráfego de rede. Isto é, intercepta os pacotes de dados, enviados e recebidos, copia-os e apresenta-os com detalhe humanamente legível na sua interface gráfica.

Para instalarmos o Wireshark, corremos o seu instalador, carregamos "*Next*" e vamos para o menu da licença de uso. Premimos "*Noted*" caso aceitemos. "*Next*" novamente e encontramo-nos no menu de componentes. Deixei no estado padrão e simplesmente pressionei "*Next*". Neste novo menu, podemos escolher criar atalhos ou lançar a aplicação assim que iniciamos a sessão no nosso dispositivo. Ao carregarmos "*Next*" somos direcionados para o menu onde escolhemos a pasta onde o queremos instalar. Carregamos novamente no botão e temos o menu onde o Wireshark nos informa que precisa de uma biblioteca para capturar pacotes de tráfego de dados. Carregando no "*Next*", finalmente a aplicação começa a ser instalada.

Ao acabar, carregamos novamente no botão "*Next*" e chegamos ao último menu onde carregamos "*Finish*". Quando corremos a aplicação, chegamos à interface gráfica. (Figura 4)

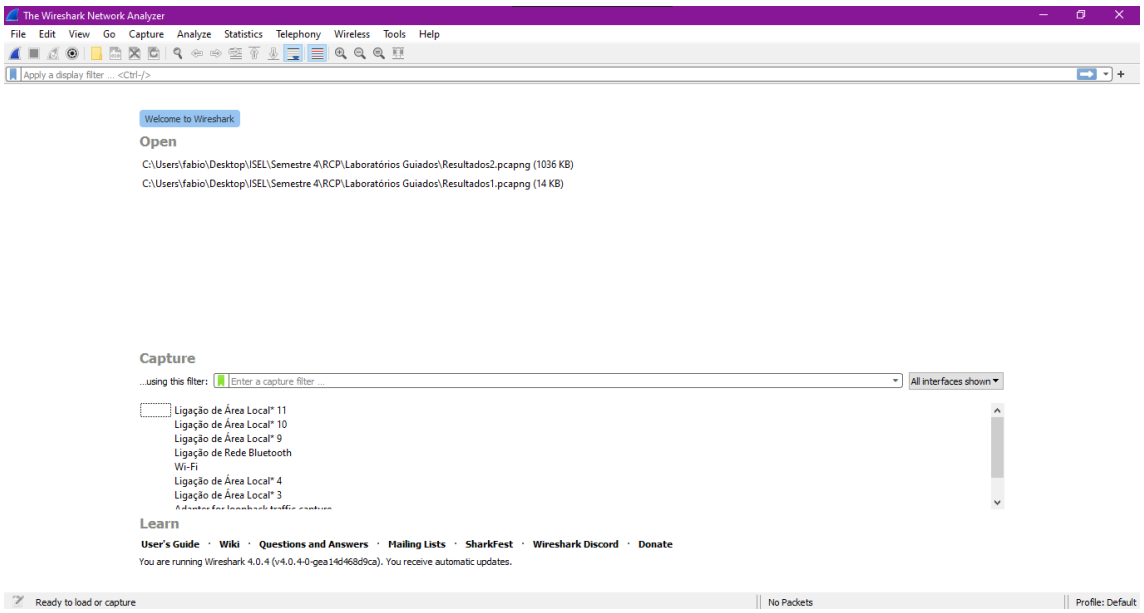

Figura 4 - Wireshark, Menu Inicial

6. Acesso por Outro Dispositivo

Passamos então a aceder ao Servidor Apache instalado previamente, a partir de outra máquina. Para isto, necessitamos de obter o IP da máquina onde se encontra o Servidor e, a partir de um browser à escolha, inserir esse IP na barra de endereço e obtemos a mesma página que obtivemos previamente.

Usando o comando *ipconfig* em ambas as máquinas, foi possível observar que a máquina onde se encontra o Servidor possui o IP 192.168.1.9. Na outra máquina é identificado o IP 192.168.1.100. (Figura 5)

Figura 5 - Servidor Apache, Acesso por outra Máquina

Com o auxílio do Wireshark, podemos inspecionar a troca de mensagens entre o Servidor e máquina de fora. (Figura 6)

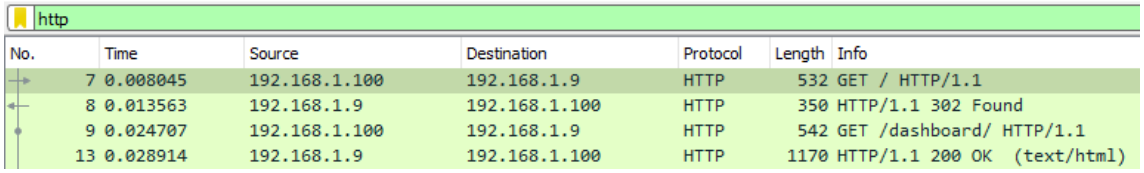

Figura 6 - Wireshark, Análise de Comunicação entre Máquina e Servidor

Como é possível observar, a origem do pedido vem do IP 192.168.1.100, a máquina que não possui o Servidor, para o IP destino 192.168.1.9, a máquina com o Servidor.

Como também é possível observar, o pedido GET é seguido do caracter "/", isto é a raiz do Servidor. Ao qual o Servidor responde com o código HTTP 302 Found, o que significa que o Servidor enviou a nova localização que o browser pretendeu aceder. De seguida, o browser automaticamente gerou um novo pedido com a localização do Servidor atualizada, ao que é respondido com o código HTTP 200 OK. Assim, foi possível aceder ao Servidor.

Alterando um pouco o endereço no browser, conseguimos obter o código HTTP 404 Not Found. Como também é possível observar, não só conseguimos ver o IP que acedemos como também o porto, neste caso, 80. (Figura 7)

Figura 7 - Outra Máquina, Acesso ao Servidor, Erro HTTP 404 Not Found

Ao observamos esta interação no Wireshark, obtemos as informações coerentes com o resultado. (Figura 8)

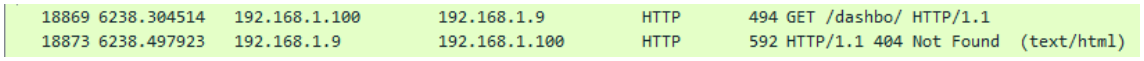

Figura 8 - Wireshark, Código HTTP 404 Not Found

7. Aplicação

A aplicação para o envio dos diversos pedidos HTTP foi desenvolvida em Python. Foi criado um pequeno menu de interação com o utilizador, de forma a ser recebido o código desejado, assim como terminar a aplicação. (Figura 9)

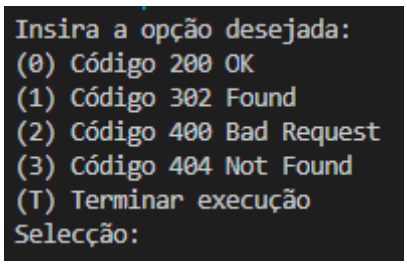

Figura 9 - Aplicação, em Execução

Ao inserir o número "0", a aplicação envia um pedido HTTP GET bem estruturado, o que permite que o Servidor interprete essa mensagem e a devolva o código 200 OK. (Figura 10)

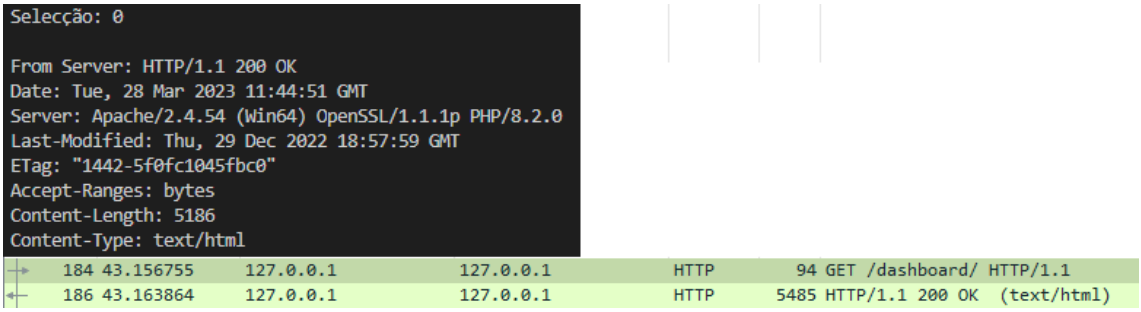

Figura 10 - Aplicação, Wireshark, Código HTTP 200 OK

Com o número 1, é enviado um pedido HTTP GET apenas com a raiz do Servidor, ou seja, apenas com o caracter "/". Este é igual ao que foi enviado pelo browser na outra máquina. Logo, logicamente, a resposta será o mesmo código HTTP 302 Found. Embora o recurso tenha sido de facto encontrado, não existe um recurso específico que possa ser devolvido. (Figura 11)

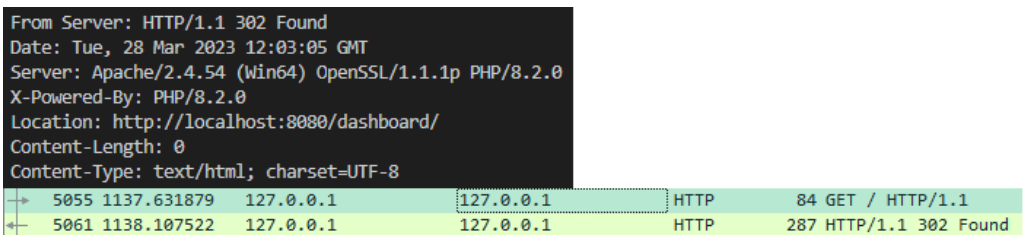

Figura 11 - Aplicação, Wireshark, Código HTTP 302 Found

Na opção 2, foi criado um pedido HTTP GET extremamente detalhado. que gera o código HTTP 400 Bad Request. Pelo meu entender, o Servidor está configurado para processar parâmetros específicos do protocolo HTTP. Se fornecer outros parâmetros que não necessita, nem os confirma, logo, considera-o um mau pedido. (Figura 12)

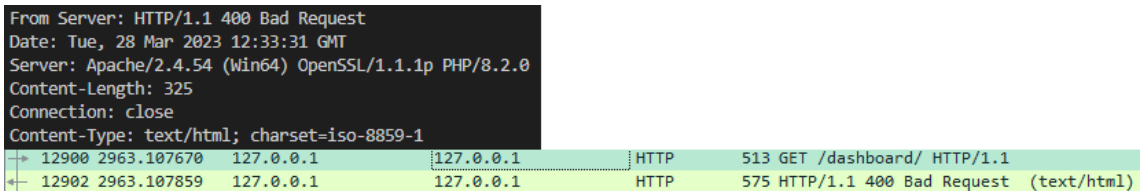

Figura 12 - Aplicação, Wireshark, Código HTTP 400 Bad Request

Com o número 3, o pedido executado vai possuir um caminho que não existe, logo, a resposta será o código HTTP 404 Not Found. O que já era esperado, dado que conseguimos obter o mesmo código, previamente, na outra máquina, ao inserirmos um caminho não existente. Isto porque o recurso não existe, como implicado pelo texto do código. (Figura 13)

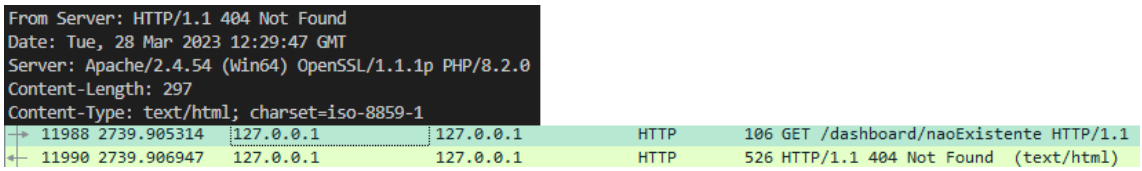

Figura 13 - Aplicação, Wireshark, Código HTP 404 Not Found

Por fim, inserindo o caracter “T”, a aplicação é terminada.

Foi também procurado obter o código HTTP 505 HTTP Version Not Supported. Para o obter, bastava enviarmos uma versão do protocolo HTTP diferente da que o Servidor estava configurado e este código seria enviado como resposta. A meu entender, a própria configuração do Servidor protege-o desta eventualidade, forçando o protocolo HTTP a ser da versão que o próprio está configurado.

Todos os pedidos desenvolvidos para obter estes códigos respostas vão estar disponíveis para leitura posteriormente no próximo capítulo.

8. Implementação da Aplicação

```
from socket import *

class Client:

    #Define o IP e o Porto do Servidor
    __serverAdress = "127.0.0.1"
    __serverPort = 80

    #Variável de Controlo da Aplicação
    __aCorrer = True

    #Opções válidas para o programa
    __opcoes = ['t', '0', '1', '2', '3']

    #Verificar se o Input dado pelo Utilizador pertence às opções
válidas
    def validateInput(self, input):

        for opcao in self.__opcoes:
            if(input == opcao):
                return True

        return False

    def main(self):
        while(self.__aCorrer):

            menu =  "Insira a opção desejada:\n" \
                    "(0) Código 200 OK\n" \
                    "(1) Código 302 Found\n" \
                    "(2) Código 400 Bad Request\n" \
                    "(3) Código 404 Not Found\n" \
                    "(T) Terminar execução"

            print(menu)

            #Obter o Input do Utilizador
            userInput = input("Selecção: ").lower()
            print()

            #Opção Inválida
            if(not self.validateInput(userInput)):
                print("Opção Inválida")
                print()
            else:
                #Terminar o Programa
                if(userInput == "t"):
```

```
                    return

                #Create Socket
                clientSocket = socket(AF_INET, SOCK_STREAM)

                #Connect to Socket
                clientSocket.connect((self.__serverAdress,
self.__serverPort))

                sentence = ""

                #200 OK
                if userInput == "0":
                    sentence = "GET /dashboard/ HTTP/1.1\r\n" \
                    "Host: localhost:8080\r\n\r\n"
                #302 Found
                elif userInput == "1":
                    sentence = "GET / HTTP/1.1\r\n" \
                    "Host: localhost:8080\r\n\r\n"
                #400 Bad Request
                elif userInput == "2":
                    sentence = "GET /dashboard/ HTTP/1.1\r\n
HOST:127.0.0.1\r\n" \
                    "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) " \
                    "AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.0.0 " \
                    "Safari/537.36 Edg/111.0.1661.41\r\n" \
                    "Accept:
text/html,application/xhtml+xml,application/xml;" \
                    "q=0.9,image/webp,image/apng,*/*;q=0.8,application
/signed-exchange;v=b3;q=0.7\r\n" \
                    "Accept-Language: pt-PT,pt;q=0.9,pt-
BR;q=0.8,en;q=0.7,en-US;q=0.6,en-GB;q=0.5,da;q=0.4\r\n" \
                    "Accept-Encoding: gzip, deflate, br\r\n" \
                    "Connection: keep-alive\r\n\r\n"
                #404 Not Found
                elif userInput == "3":
                    sentence = "GET /dashboard/naoExistente
HTTP/1.1\r\n" \
                    "Host: localhost:8080\r\n\r\n"

                #Envio da mensagem codificada
                clientSocket.send(sentence.encode())

                #Mensagem Recebida, enviada pelo Servidor
                modifiedSentence = clientSocket.recv(1024)

                #Imprimir mensagem do Servidor, descodificada
```

```
                print ('From Server:', modifiedSentence.decode())

                #Fecho do Socket
                clientSocket.close()
                print()


#Correr o código
cliente = Client()
cliente.main()
```

9. Conclusões

Uma das conclusões obtidas ao desenvolver este trabalho foi a quantidade abismal de dados que existem e que não são visíveis aos olhos do utilizador. Para cada pequena camada, um novo troço é adicionado à mensagem que, por sua vez, ao ser recebido, tem de ser partido da forma correta. Com isto, é possível ter uma noção do quanto o browser facilita o acesso ao utilizador, pois isto acontece tudo de forma invisível ao utilizador, sem ser necessária a sua intervenção.

Foi também possível perceber que é possível salvaguardar a ligação do Cliente ao Servidor caso o protocolo HTTP não seja compatível a partir das configurações do Servidor. Este pode forçar a versão do protocolo, o que, assim, não impede o utilizador de aceder ao serviço.

Foi obtida uma compreensão da distinção entre mensagens pedido e mensagens resposta, assim como a organização de cada uma. Para diferentes pedidos, existem diferentes ações e são os protocolos que permitem esta interação funcional, fluída e, de certa forma, universal.

10. Bibliografia

[1] L. Pires, Slides, “Computer Networks: Chapter2”, Março 2023

[2] “Wireshark”, [Online]. “https://www.wireshark.org/”.

[3] “Protocolo HTTP”, [Online]. “https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol”.

[4] “RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1”, [Online]. “https://www.rfc-editor.org/rfc/rfc2616”.

[5] “Layered Internet Protocol Stack”, [Online]. “https://www.w3.org/People/Frystyk/thesis/TcpIp.html”.

[6] “TCP/IP”, [Online]. “https://pt.wikipedia.org/wiki/TCP/IP”.

[7] “Servidor Web Apache”, [Online]. “https://www.apachefriends.org/”.